APPENDIX A

Server Data Structures Design Specification

Table of Contents

1 Overview of Server Structures

The server consists of the following structures:

1.1 Configuration

The Configuration file defines where all the pieces are stored and how to access them. This allows Selva to access these components by name, without regard to where they are, making for easier caching.

1.2 Identity

The Identity of the server is contained in a file called Identity.xml, which contains information that identifies the enterprise within which the server operates. It also contains static global information, such as JDBC URLs, ADO Connection Strings, HTTP addresses, SMTP addresses, and other static information that may be required by specific applications.

In addition to the parts of the Identity that are required (or expected) by the server, there is a <userDefined> element that contains any user defined elements that an application may require.

1.3 Personality

The Personality of the server is contained in a file called Personality.xml, which contains information that identifies specific XML messages or SOAP function calls that can be supported by the server, and routes these to the appropriate workflow to handle these.

1.4 Messages

The Messages that the server returns for error conditions are defined in a Messages.xml file.

1.5 Behavior

The Behavior of the server for any specific type of message or function call, is defined by a "behavior" file, which is an XML file which defines the actions to be taken by the server to fulfill the request that was made.

2 Configuration

```
<selva:Configuration
        xmlns:selva="http://www.gaexpress.com/Selva/Configuration">
    <diagnostics logFile="out|err|C:\directorypath\for\logfile"
                 DASL="0|1|2"
                 RASL="0|1|2"
                 server="0|1|2"
                 XSLT="0|1|2"
                 workingMemory="0|1|2"
                 behavior="0|1|2"
                 database="0|1|2" />

    <server basePath="c:\Proj\Selva\RunArea\"
            slash="\"
            personality="Personality.xml"
            identity="Identity.xml"
            messages="Messages.xml"
            hostName="rhlaptop2k.libertyodbc.com"
            protocol="http"
            socket="50080"
            threads="1"
            adminProtocol="http"
            adminSocket="50081"/>

    <DASL DBRulesPath="DBRules\" />
    <RASL RPCRulesPath="RPCRules\" />
    <XSLT XSLTPath="XSLTFiles\" />
    <behavior behaviorsPath=Behaviors\" />
    <drivers>
        <driver type="JDBC"
                className="sun.jdbc.odbc.JdbcOdbcDriver" />
    </drivers>
</selva:Configuration>
```

2.1 Configuration

This is the document element of the server configuration file.

2.2 diagnostics

Diagnostics all have 3 levels:

- 0=None
- 1=Normal
- 2=Detailed

2.2.1 logFile

This is the path to the directory where the log file will be created. The log file is where log and diagnostic information is to be placed. The log file name itself generated by the server and will be globally unique and have a .log extension.

If a value of "out" or "err" is specified instead of a directory path, then log output goes to System.out or System.err, respectively.

2.2.2 DASL

DASL diagnostics track actions done by DASL. Normal means that commands and return values are dumped to the log file, along with indexes into the node lists returned.

2.2.3 RASL

RASL diagnostics track actions done by RASL. Normal means that input values and output values are dumped.

2.2.4 server

This indicates diagnostics that are relevant to the server's operation.

2.2.5 XSLT

This tracks transformations.

2.2.6 workingMemory

This tracks the contents of workingMemory at various points.

2.2.7 behavior

This tracks the order in which actions are fired in behaviors, and which tests in a criteria element caused the execution to occur.

2.2.8 database

This tracks database I/O.

2.3 server

This is a container element that contains configuration information that is used by the server to determine where to put things or how to run.

2.3.1 basePath

This attribute defines the base path for all other attribute that specify a location or file. If the reference is not absolute, this is appended to the front of the other attribute before retrieving the file.

2.3.2 slash

This indicates the path separator character (which slash to use) for file system access.

2.3.3 personality

This attribute defines the path and filename of the personality XML file.

2.3.4 identity

This attribute defines the path and filename of the identity XML file.

2.3.5 messages

This attribute defines the path and filename of the messages XML file.

2.3.6 hostName

This is the hostname that is listening. Not always possible to get in a standard way in Java, so we hardwire it here. The IDE needs this as well.

2.3.7 protocol

This is the protocol that this server listens with. Current option is "http", "https" future option.

2.3.8 socket

This is the socket on which the server listens for HTTP requests and SOAP messages.

2.3.9 threads

This is the number of threads to launch for the server.

2.3.10 adminProtcol (FUTURE)

This is the protocol that this server listens on the administrative socket with for privileged administrative messages. Current option is "http", "https" future option.

2.3.11 adminSocket (FUTURE)

This is the socket on which the server listens for privileged administrative messages.

2.4 DASL

This element is a container for attributes that define where to find components for DASL processing.

2.4.1 DBRulesPath

This is the path at which all DBRules files will be found.

2.5 RASL

This element is a container for attributes that define where to find components for RASL processing.

2.5.1 RPCRulesPath

This is the path at which RPCRules files will be found.

2.6 XSLT

This element contains the attribute that points to where XSLT files are retrieved from.

2.6.1 XSLTPath

This provides a path to where XSLT files are read from.

2.7 behavior

This is a container element for attributes that define where to find behavior files.

2.7.1 behaviorsPath

This is the path at which all behavior files can be found.

2.8 drivers

A container for drivers (like JDBC drivers) that need to be initialized

2.9 driver

A container for a specific driver.

2.9.1 type

JDBC is the only relevant type, for now.

2.9.2 className

The name of the class to initialize with Class.forName("...")

e.g.: "sun.jdbc.odbc.JdbcOdbcDriver" is the JDBC-ODBC bridge driver that lets us call ODBC drivers.

3 Identity

```
<selva:Identity xmlns:selva="http://www.gaexpress.com/Selva/Identity">
   <locale>localIdentifier</locale>
   <enterpriseInfo>
      <companyName>XYZ company</companyName>
      <companyAddress>1234 5th Street</companyAddress>
      <companyAddress2/>
      <companyCity>Irvine</companyCity>
      <companyState>CA</companyState>
      <companyZip>98765-4321</companyZip>
      <companyContact>Jane Christie</companyContact>
      <companyPhone>(949) 250-4800</companyPhone>
      <companyFax>(949) 123-4567</companyFax>
      <companyWeb>http://www.gaexpress.com</companyWeb>
      <companyEmail>info@gaexpress.com</companyEmail>
      <divisionName/>
   </enterpriseInfo>
   <SMTPTargets>
      <SMTPTarget name="targetName"
                  server="mail.gaexpress.com"
                  account="sales@gaexpress.com"
                  userID="sales" />
   </SMTPTargets>
   <URLs>
      <URL name="URLName"
           type="WSDL|POST|SOAP|FILE"
           URLString="URLToPostTo"
           method="basic"
           userID="userId"
           password="password"
           domain="domain"
           authXPath="//authentication[role='admin']" />
   </URLs>
   <connections>
      <connection name="ConnectionName"
                  type="JDBC|ODBC|ADO|DIR"
                  connectString="ADOorODBCConnectionString"
                  jdbcURL="JDBCURL"
                  userID="userId"
                  password="password"/>
   </connections>
   <authentications>
      <authentication role="admin"
                      method="basic"
                      userID="foo"
                      password="bar"
                      domain="myDomain" />
   </authentications>
   <userDefined />
</selva:Identity>
```

3.1 Identity

This is the document element for the Identity file.

3.2 locale

This element is used to provide an identifier that indicates the locale, in order to provide localization information. We may modify this to properly handle locale-specific processing.

3.3 enterpriseInfo

This section contains information about the enterprise that the server represents and in which is operates. We will not comment on sub elements, as they are considered self-evident.

4 SMTPTargets

This is the first in a series of sections that describe resources that can be used to send or receive information from other entities both within the organization and outside of it. The particular focus of this section is SMTP (email) servers and accounts to which notifications or data can be sent.

4.1.1 name

This is used to identify the SMTP target both for diagnostics, and to make the XPath easier and more readable.

4.1.2 server

This is the URL for the SMTP server to which we will send the data.

4.1.3 account

This is the account at that server where the messages are stored.

4.1.4 userID

This is the user id to which we will send the message.

5 URLs

This is the next section, and is used to define URLs to which POST requests can be sent, both for SOAP and for POSTing requests or responses to partners. The intent is that this would be used to reference global values that all applications might use, like the return address to tell a third party partner to use to post to us, for instance, or the URL of another division within the same enterprise, or a key partner.

5.1 URL

This is the container element for an individual URL.

5.1.1 name

This is the name of the URL, which is used mostly to allow simply XPath predicates to reference this URL by it's @name attribute.

5.1.2 type

This is a convenience attribute that is used to indicate what the URL is used for. Two possibilities are WSDL (Web Service Description Language) queries, or POSTing commands to a partner.

5.1.3 URLString

This would be the actual URL to use.

5.1.4 method

This is the method of authentication to use (e.g. "basic").

5.1.5 userID

If authentication is required, this would be the user id to be used.

5.1.6 password

If authentication is required, this would be the password to be used.

5.1.7 domain

If authentication is required, and if this applies, this would be the domain to be used for authentication.

5.1.8 authXPath

This would be the XPath applied to the Identity file (this file) that identifies and authentication element that contains information to be used for authentication purposes.

6 connections

This section exists to define DBMS resources, which can be accessed through JDBC, ODBC or ADO.

6.1 connection

This is the container for an individual connection

6.1.1 name

This is a convenient name to allow XPath predicates to point to this connection by name.

6.1.2 type

This is a simple type string.

6.1.3 connectString

This is either the ODBC connection string, the ADO connection string, or the path to the directory.

6.1.4 jdbcURL

This is the jdbc URL to be used.

6.1.5 userID

This is the user id to authenticate the connection with.

6.1.6 password

This is the password to authenticate the connection with.

7 authentications

Container for all authentications that will be used.

7.1 authentication

Wrapper for an individual authentication.

7.1.1 role

Name by which an authentication is accessed.
The XPath to access the authentication for the "admin" role, would be:

```
/Identity/authentications/authentication[@role="admin"]
```

7.1.2 method

This is the authentication method that this role is intended to be used with.

7.1.3 userID

The user name or id to use.

7.1.4 password

The password associated with the user name.

7.1.5 domain

Optional domain for NTLM (NT LAN Manager) or BASIC authentication to an NT or Windows 2000 server.

8 Personality

```
<selva:Personality
            xmlns:selva="http://www.gaexpress.com/Selva/Personality">

   <businessDocuments>
      <businessDocument name="RosettaNetPORequest"
                        type="XML|Java|EJB|COM|Resume"
                        objectId="ProgID|JavaClass|Bean"
                        businessProcess="businessProcessName"
                        documentBehavior="RosettanetPORequest.xml"
                        connectionXPath="XPathToConnection"
                        GUIDSource="documentName"
                        GUIDXPath="XPathToGUID"
                        newInputDocumentName="ResponseDocument2">
         <criteria>
            <testBatch>
               <test testSourceName="inputDocument"
                     XPath="/Pip3A4PurchaseOrderRequest"/>
                  Other test's here
            </testBatch>
            other testBatch's here
         </criteria>
      </businessDocument>
      other businessDocument's here
   </businessDocuments>

   <businessProcesses>
      <businessProcess  name="PurchaseOrder"
                        processBehavior="behaviorfile.xml" />
      other businessProcess's here
   </businessProcesses>

   <businessMethods>
      <businessMethod name="myFunction"
                      WSDLFile="WSDLFiles\OurWSDLFile.xml"
                      type="Java|EJB|COM|XML"
                      pathInfo="GETorPOSTPathInfo"
                      localObjectName=" ProgID|JavaClass|Bean" />
      other businessMethod's here
   </businessMethods>

</selva:Personality>
```

8.1 Personality

This is the document element for the Personality file.

8.2 businessDocuments

This is the container document for all businessDocument descriptions. There is only ever one of these in a Personality file. This is one of two key sections in the Personality file. This one is responsible for dealing with POSTed (or emailed, or FTPed) XML documents. This deals with document-based messaging.

8.3 businessDocument

This is the container element for a businessDocument description. There can be many of these in the Personality file. These are used to identify a business document that has been received, identify (optionally) how to transform it (i.e. which *document behavior to apply to it*), and where to send it for processing (i.e. which *process behavior* to apply to it).

8.3.1 name

This is a handy name for referring to the business process that is used to handle this particular type of document. It is here more for diagnostic purposes than anything else.

8.3.2 type

This attribute allows you to specify what type of behavior we are dealing with. The default value for this is "XML". It could be

8.3.3 objectId

This is either the ProgID for a COM object, or the Java class name, or the bean name to be called. The relevant programming object must have a method with the definition:

Boolean IsMyType(String InputDocumentXML)

It returns True to indicate that this document is indeed one that must be processed. Also, you must have the following method that returns the XML for the working document:

String Convert(String InputDocumentXML)

8.3.4 businessProcess

This is the name of the business process that implements the functionality to handle this document.

8.3.5 documentBehavior

This is the path to the behavior file that describes how to process the received document in preparation for the business process. Normally it consists of three parts:

- An optional validation
- A transformation to the working document format

- A call to the process behavior

Alternately, it can be the special string "*Resume", in which case we resume an existing transaction and complete it.

8.4 Resuming Suspended Transactions

The following attributes exist for the case where the behavior is "*Resume".

8.4.1 connectionXPath

This is the XPath to a connection (in the connections section of the Identity file) that is used to save/retrieve suspended transactions.

8.4.2 GUIDSource

This is the name of the working document from which to get the GUID.

8.4.3 GUIDXPath

This is the XPath (in the InputDocument) of the location that contains a GUID for reconstituting the suspended transaction.

8.4.4 newInputDocumentName

When a transaction was suspended, it had it's own "InputDocument" in the Working Memory. The new transaction also has an InputDocument, so we use this attribute to rename the current InputDocument, so that the reconstituted document can have still have it's original "InputDocument". This maintains the integrity of the original transaction.

8.5 businessProcesses

This is the container for a list of business processes that revolve around your LOB system, like a "Price Check" or "Inventory Request" or "Purchase Order Request".

8.6 businessProcess

Each of these represents a business process that revolves around your LOB system.

8.6.1 name

This is a friendly name for the business process.

8.6.2 processBehavior

This is the behavior file that defines how to work with the canonical form of the document.

8.7 businessMethods

This is the container element for the section that deals with method calls to the Selva server over SOAP.

8.8 businessMethod

This is a container element for a specific test for a business method that will handle the incoming request. This section is processed somewhat differently than the businessDocument sections, in that the real identifier is the name attribute.

These are used for both providing metadata to clients that request it and for handling requests from clients.

8.8.1 name

This mandatory attribute really serves two purposes. It uniquely identifies this business method from any other business method in the Personality file and it is the name of the Business Method (a.k.a. Web Service, a.k.a. SOAP Method) that the Selva Server will now provide

8.8.2 pathInfo

If this attribute exists, it is used to match against the POSTed path information that was received from the client.

8.8.3 WSDLFile

This is the path to the Web Services Description Language (WSDL) XML file. This file is used to provide metadata to a calling application.

8.8.4 type

This indicates what type of internal implementation we are dealing with.

8.8.5 localObjectName

This is the ProgID, Java class name, or bean to be called to process this message. This is irrelevant if the type attribute has the value 'XML'.

8.8.6 translator

This would be the XSLT file that would translate the incoming SOAP message into a WorkingDocument in the Working Memory, and would be used if the type attribute had the value 'XML'.

8.9 Handling Criteria in Personality files

The personality file uses Criteria to identify if a specific document that has been input is one that the current businessDocument element describes processing for.

9 Messages

The following is a portion of the Messages.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Messages [
      <!ELEMENT default (#PCDATA)>
      <!ATTLIST localeSpecific localeId CDATA #REQUIRED >
      <!ELEMENT localeSpecific (#PCDATA)>
      <!ATTLIST message Id ID #REQUIRED >
      <!ELEMENT message (default,localeSpecific) >
      <!ELEMENT Messages (#PCDATA | message)* >
]>
<selva:Messages xmlns:selva="http://www.gaexpress.com/Selva/Messages">
   <message Id="M1">
      <default>Posted message is not well-formed XML</default>
      <localeSpecific localeId="localeID">
         locale specific text goes here
      </localeSpecific>
   </message>
         More messages in here
   <message Id="M29">
      <default>SOAP Method call failed</default>
      <localeSpecific localeId="localeID">
         locale specific text goes here
      </localeSpecific>
   </message>
</selva:Messages>
```

9.1 DOCTYPE

In order to speed processing, we use an inline DTD to specify that the Id attribute is of type ID, which allows us to access it by it's unique value.

9.2 Messages

This is the document element for the Messages.xml file.

9.3 message

This is the container for a single message. It is referenced by looking up it's Id attribute.

9.3.1 Id

This is the Id that allows us to do a quick lookup into this document.

9.4 default

When no locale is specified, this is the default message to be returned.

9.5 localeSpecific

When a localeID value is specified in the Identity file, you can look up the localeSpecific element with the localeID attribute that matches the localeID from the Identity file.

9.5.1 localeID

This is the attribute used to match a specific localeSpecific element to your localeID.

10 Behavior

Handling of the Behavior file is dealt with in a separate document

11 Handling Criteria

In both the Personality file and in the Behavior file, we use a combination of working memory and XPath notation to allow you to specify whether certain conditions exist to allow a path to be followed. The follow section describes how this works.

11.1 criteria

This identifies that a criteria test must be applied to validate whether the action or personality choice will be taken. This is the container for this test.

11.2 testBatch

This is the container for a single "OR" test. These are applied until one testBatch element returns True.

11.3 test

These tests are "AND"ed, effectively applied until one fails. If all pass, we return True, if any fails, no further testing is done and we return False.

11.3.1 testSourceName

This is the name of the input document against which to apply the tests. Note that while XPath allows you to do OR and AND logic within a single document, it does not have a facility for crossing document boundaries, hence the need for our more elaborate test structure.

11.3.2 XPath

This is the XPath to apply to the source document in order to determine whether this part of the test returns True or False. If any nodes are returned, the test returns True, otherwise it returns False.

12 Other thoughts

The following items need to be discussed:

- Schema for Identity, Personality and Behavior files
- Think through the authentication issues
- Deal with namespace issues

The **testSourceName** attribute of the <test> element in <testBatch> in <criteria> in <businessDocument> should be renamed to **source.**

13 Server Administration

13.1 Administration specifications

13.1.1 Personality.xml file

This should contain the following lines to enable the use of Selva Administration Commands

```
<businessProcesses>

        <businessProcess name="SelvaAdminProcess" processBehavior="SelvaAdminProcess.xml"/>

</businessProcesses>

<businessDocuments>

    <businessDocument   name="SelvaAdminDocument"
                        type="Admin"
                        businessProcess="SelvaAdminProcess" >
                        documentBehavior="CopyDocument.xml">
        <criteria>
            <testBatch>
                <test testSourceName="inputDocument" XPath="/SelvaAdminCommand" />
            </testBatch>
        </criteria>
    </businessDocument>

        <criteria>
            <testBatch>
                <test testSourceName="inputDocument" XPath="/SelvaAdminCommand"/>
            </testBatch>
        </criteria>
    </businessDocument>

</businessDocuments>
```

13.1.2 SelvaAdminProcess.xml file

This will contain

```
<?xml version="1.0" encoding="UTF-8"?>
<selva:Behavior xmlns:selva="http://www.gaexpress.com/Selva/Behavior">
    <description>
        This processes incoming Selva Administration command messages, which have been copied into
the
        workingDocument document in workingMemory by the CopyDocument.xml document behavior of
    the     SelvaAdminDocument businessDocument
    </description>
    <action name="DoAdmin" type="Admin" />
</selva:Behavior>
```

13.2 Administrative Messages

The Administrative Messages will have the form

```
<selva:SelvaAdminCommand
      xmlns:selva="http://www.gaexpress.com/Selva/Admin">
   <description>
      What this Administrative command does
   </description>
   <command type="continue|stop|refresh"
            parameters="xxxx">
</selva:SelvaAdminCommand >
```

13.2.1 description

This is ignored by the server and is simply for descriptive purposes.

13.2.2 command

This is the single container for the administrative command.

13.2.3 type

The "continue' command is just for debugging purposes.

The "stop" command will stop all the server threads after they have completed their current transaction.

//PROBLEM How do we stop threads which are waiting on an accept?

The "refresh" command will cause the configuration to be reloaded each thread. (The exact definition of what 'reloading' means is still to be determined.)

13.2.4 parameters

This is for future use and is currently undefined, but clearly it will be used to provide parameters to the commands.

Selva Action Tests

Table of Contents

1 Common Behavior Definition

The following shows the common elements of the Behavior file. All action elements can contain these elements.

```
<Behavior xmlns="http://www.gaexpress.com/Selva/Behavior">
   <description>
      What this behavior is all about
   </description>
   <action name="xxxxxxxxxx"
           type="xxxx">
      <description>
         What this action is all about…
      </description>
      <criteria>
         See Server Structures document for description of criteria
      </criteria>
   </action>
    other actions here
</Behavior>
```

1.1 Criteria Tests

- Test that the OR logic in the testBatch elements works correctly.
- Test that the AND logic in the test elements works correctly.

2 XSLTTransformer action

Below is a sample fragment of a series of XSLTTransformer type actions:

```
<action name="AckPOReq"
        type="XSLTTransformer"
        XSLTFile="POResponseAck.xsl"
        source="*now"
        target="HTTPResponseDocument">
   <description>
      Acknowledges the receipt of a PO Request by adding the current
      date and time to the document which will be sent back via
      HTTP.
   </description>
   <criteria>
   </criteria>
</action>

<action name="AddGUIDtoPO"
        type="XSLTTransformer"
        XSLAFile="Copy.xsl"
        source="*GUID"
        target="WorkingDocument"
        targetXPath="/PurchaseOrder">
   <description>
      Adds a GUID element (which is a unique reference) to the
      current working document under the PurchaseOrder element.
   </description>
   <criteria>
   </criteria>
</action>

<action name="InformNotInStock"
        type="XSLTTransformer"
        XSLTFile="CopyNotInStock.xsl"
        source="WorkingDocument"
        target="WorkingDocument"
        targetXPath="/PurchaseOrder">
   <description>
      Adds an indication that the product is not in stock into the
      current working document under the PurchaseOrder element.
   </description>
   <criteria>
   </criteria>
</action>

<action name="ShipByFedex"
        type="XSLTTransformer"
        XSLTFile="Copy.xsl"
        source="*String"
        target="WorkingDocument"
        targetXPath="/PurchaseOrder">
   <description>
   Copies the instructions to ship the order by FedEx from the
   SourceString element below
```

```
    </description>
    <criteria>
    </criteria>

    <sourceString>
       <![CDATA[
          <ShippingInfo>Please ship this via FedEx </ShippingInfo>
       ]]>
    </sourceString>
 </action>
```

2.1 source

Using the simple Copy.xsl XSLT file, we should test that the source values of 'inputDocument', '*String', '*now' and '*GUID' work as documented.

2.1.1 source = "*GUID"

In this case, we construct, as the input document, a simple XML file with the structure:

```
<GUID>guidValue</GUID>
```

Where guidValue is a GUID value associated with the current transaction. The Director creates this value and places it into the Behavior file as an attribute of the document element, called "GUID".

2.1.2 source = "*now"

In this case, we construct, as the input document, a simple XML file with the structure:

```
<now>yyyy-mm-dd hh:mm:ss</now>
```

2.1.3 source = "*String"

In this case, the sourceString element will contain the XML to be processed, as either a text string or a CDATA section.

2.2 sourceString

This value is only valid if the source attribute is "*String". In this case, this is the XML string and is loaded into a DOM tree for transformation. In the last example above, this copies a literal string into the target as a subtree under the node identified by the targetXPath attribute.

2.3 target and targetXPath

Ensure that the transformer works when a targetXPath is present and when a targetXPath is missing.
This is the name of the WorkingMemory object into which the output is to be put.

3 HTTPResponse action

Below is a sample fragment of an HTTPResponse type action:

```
<action name="xxxxxxx"
        type="HTTPResponse"
        source="*String|workingMemoryName">
   <description>
      What this action is all about…
   </description>
   <criteria>
      See Server Structures document for description of criteria
   </criteria>
   <sourceString>
      <![CDATA[
         <myRoot>somedata</myRoot>
      ]]>
   </sourceString>
</action>
```

3.1 source

Test that both '*String' and inputDocument work.

4 Suspend action

Below is a sample fragment of a Suspend type action:

```
<action name="xxxxxxx"
        type="Suspend"
        connection="/Identity/connections/connection[@name='xxx']"
        GUIDSource="WorkingDocument"
        GUIDXPath="/PurchaseOrder/GUID">
   <description>
      What this action is all about…
   </description>
   <criteria>
      See Server Structures document for description of criteria
   </criteria>
</action
```

Note this must be tested in conjunction with a resuming transaction.

4.1 connection

Test with both a directory and a JDBC type connection resource.

4.2 GUIDSource and GUIDXPath

Test both with these values and without them.

5 HTTPPOST action

Below is a sample fragment of an HTTPResponse type action:

```
<action name="xxxxxxx"
        type="HTTPPOST"
        source="*String|OutputDocument1"
        identityURLXPath="/Identity/URLs/URL[@name='URLName']"
        URLSource="workingDocument"
        URLXPath="/Supplier/@URL"
        URLString="http://myserver.com/posthere.cgi"
        target="ResponseDocument1"
        targetXPath=""
        authXPath="//authentication[@role='admin']"
        method="basic"
        userId="foo"
        password="bar"
        domain="myDomain">
   <description>
      What this action is all about…
   </description>
   <criteria>
      See Server Structures document for description of criteria
   </criteria>
   <sourceString>
      <![CDATA[
         <myRoot>somedata</myRoot>
      ]]>
   </sourceString>
</action>
```

5.1 source

Test with both '*String' and 'inputDocument'.

5.2 identityURLXPath

Test with an identityURLXPath to get the URL.

5.3 URLSource and URLXPath

Test with URLSource and URLXPath to get the URL.
The first node returned by the node-list from the XPath is evaluated, and if it is an attribute, its value is taken; otherwise (for an element), its nodeValue is taken.

5.4 URLString

Test with URLString to get the URL.

-

5.5 target and targetXPath

Test both with and without a targetXPath value.

5.6 Authentication values

Do not test the following at this time:

- authXPath
- method
- userId
- password
- domain

5.7 Error Handling for HTTPPOST action

If an HTTP error is encountered, HTTPResponse document fragment will look as follows:

```
<HTTPError number="500"
           URL="URLPostedTo">
   <dataPosted>
         Content of POST
   </dataPosted>
   <response>
      <![CDATA[
         Content of response
      ]]>
   </response>
</HTTPError>
```

Test against a server that is down, and an invalid URL.

6 DASL action

The DASL action is dealt with in the document that defines DASL.

7 RASL action

The RASL action is dealt with in the document that defines RASL.

8 NewBehavior action

Below is a sample fragment of a NewBehavior type action:

```
<action   name="xxxxxxx"
          type="NewBehavior"
          behavior="behaviorFile.xml"
          return="True|False"
          inputDocument="documentName"
          retainDocuments="True|False" >
   <description>
      What this action is all about…
   </description>
   <criteria>
      See Server Structures document for description of criteria
   </criteria>
</action>
```

8.1 behavior

Test calling an invalid behavior file.

8.2 return

Test with return both True and False. Ensure that it works correctly, returning or not, as specified.

8.3 inputDocument

Test that this properly renames the specified document to 'inputDocument'.

8.4 retainDocuments

Test with this set to both True and False. Ensure that when it is False, the documents are actually deleted.

9 Node action

Below is a sample fragment of a Node type action:

```
<action    name="xxxx.xx"
           type="Node"
           source="*String|documentName"
           selectSingle="True|False"
           sourceXPath="XPathToNode"
           do="delete|move|copy"
           target="documentName"
           targetXPath="XPathToTargetLocation"
           wrap="ElementNameToWrapNode(s)In"
           wrapMultiple="ElementNameToWrapMultipleNodesIn"
           textToNode="True|False"
           nodeToText="True|False" >
   <description>
      What this action is all about…
   </description>
   <criteria>
      See Server Structures document for description of criteria
   </criteria>
   <sourceString>
      <![CDATA[
         <myRoot>somedata</myRoot>
      ]]>
   </sourceString>
</action>
```

9.1 source

Test with source both "*String" and "workingDocument".

9.2 sourceXPath and selectSingle

Test with and without sourceXPath. For the sourceXPath variation(s) test both with selectSingle = False and = True.

Again, the tests are:

- no sourceXPath
- sourceXPath and selectSingle = True
- sourceXPath and selectSingle = False

9.3 do

Test the following types:

- delete
- move
- copy

9.4 target and targetXPath

Test both with and withoug targetXPath.

9.5 wrap and wrapMultiple

Test both with and without a wrap value.

9.6 textToNode and nodeToText

Test both of these variations.

10 Document action

Below is a sample fragment of a Node type action:

```
<action   name="xxxxxxx"
          type="Document"
          do="create|delete|copy|rename"
          source="*String|documentName"
          sourceXPath="XPathToNode"
          documentElementName="newDocumentElement"
          target="documentName" >
    <description>
       What this action is all about…
    </description>
    <criteria>
       See Server Structures document for description of criteria
    </criteria>
    <sourceString>
       <![CDATA[
          <myRoot>somedata</myRoot>
       ]]>
    </sourceString>
</action>
```

10.1 do

Test the following:

- create a new document with the provided documentElementName as the document element,
- create a new document from a string,
- delete a source document,
- copy a fragment of a document to a new document,
- move a fragment of a document to a new document,
- copy an entire document to a new document,
- move an entire document to a new document

11 StartLoop action

Below is a sample fragment of a StartLoop type action:

```
<action   name="xxxxxxxx"
          type="StartLoop"
          maxIterations="n">
   <description>
      What this action is all about…
   </description>
   <criteria>
      While condition goes here
   </criteria>
</action>
```

Test the following combinations:

- A loop that stops due to maxIterations being reached. Ensure that 'n' iterations are actually done.
- A loop that is skipped because the criteria doesn't match.
- A loop that can't find it's RepeatLoop throws a meaningful exception.

12 RepeatLoop action

Below is a sample fragment of a RepeatLoop type action:

```
<action   name="xxxxxxx"
          type="RepeatLoop"
          loopToName="LoopToName">
   <description>
      What this action is all about…
   </description>
   <criteria>
      While condition goes here
   </criteria>
</action>
```

12.1 loopToName

Test the case where there is no matching loopToName. (In this case this action won't be decorated with the startLoopOffset run-time attribute.

12.2 criteria

Test that the criteria properly loops back when the criteria passes.

Test that the criteria properly ends the loop when the criteria fails.

13 Validate action

Below is a sample fragment of a Validate type action:

```
<action   name="xxxxxx"
          type="Validate"
          schema="schema.xsd"
          target="workingMemoryName" >
   <description>
      What this action is all about…
   </description>
   <criteria>
      See Server Structures document for description of criteria
   </criteria>
</action>
```

Do not test at this time.

14 Custom action

Below is a sample fragment of a Custom type action:

```
<action   name="xxxxxxx"
          type="Custom"
          objectType="Java|COM|EJB"
          objectId="ProgId|package.ClassName|etc..."
          inputDocuments="*String|*|objectName1&objectName2&..."
          target="documentName"
          targetXPath="XPathToTargetLocation">
    <description>
       What this action is all about…
    </description>
    <criteria>
       See Server Structures document for description of criteria
    </criteria>
    <sourceString><![CDATA[...]]></sourceString>
</action>
```

Do not test this at this time.

15 SMTP action

Below is a sample fragment of an HTTPResponse type action:

```
<action name="xxxxxxx"
        type="SMTP"
        source="documentName|*String"
        sourceXPath="XPathToData"
        subject="documentName|*String"
        subjectXPath="XPathToData"
        subjectString="SubjectString"
        date="documentName|*now"
        dateXPath="XPathToData"
        dateString="dateOfSomeSort"
        replyto="documentName|*String"
        replytoXPath="XPathToData"
        replytoString="stringReplyTo"
        cc="documentName|*String"
        ccXPath="XPathToData"
        ccString="ccString"
        SMTPXPath="/Identity/SMPTTargets/SMTPTarget[@name='xx']"
        server="documentName"
        serverXPath="XPathToData"
        serverString="mail.mydomain.com"
        account="documentName"
        accountXPath="XPathToData"
        accountString="myAccount"
        userIdSource="documentName"
        userIdXPath="XPathToUserId"
        userId="userId" >
    <description>
       What this action is all about…
    </description>
    <criteria>
       See Server Structures document for description of criteria
    </criteria>
    <sourceString>
       <![CDATA[
          <myRoot>somedata</myRoot>
       ]]>
    </sourceString>
</action>
```

Do not test this at this time.